Design Patterns → recurrent solutions that have been abstracted as a model, which characterizes one: well documented well designed help to improve the quality of the software in term of reusability, maintainability and modularity

framework → A group of components that cooperate each other to provide a reusable architecture for application for a given domain

UML → help to simplify the complex software engineering process, its a modeling language only

UML help to represent

- static structure of an application (Class Diagrams, Object Diagrams, Component Diagrams, Deployment Diagrams)
- behaviour of an application (Use case, Sequence, Activity, collaboration, statechart)
- express model management (Packages, Subsystem, Models)

Class diagram

Describe the static structure of the system | Abstraction: Showing essential features and hiding non-essential features to the user

Encapsulation: mechanism of wrapping the data and code acting as a single unit, it is one of the fundamental concepts in O.O.P, whose application it is security

Access specifiers (+ public, - Private, # Protected, None friendly)

Abstract Class → Provide an outline of a class

A class with at least one abstract method

How can I achive abstraction in Java?

Abstract Class CO 100% Interface 100%

Interface: A concept that specifies the externally visible operations of a class, but not the actual implementation of those operations

Generalization → it is a mechanism to represent inheritance

realization → depicts the relation between an Interface and a class, that provides the actual implementation

dependency → depicts the relation between a source and a target component, when there is a change in the target component, source undergoes a change as well, but not viceversa

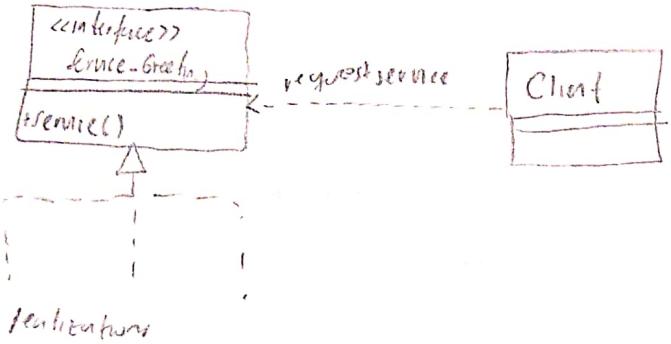
composition → express a strong ownership between classes, relation part-whole

Sequence Diagrams

helps me to depict the relationship between collaborative objects in terms of messages exchange over time

Basic Patterns

1 Interface : used to design a set of services that is implemented in classes



2 Abstract : helps me to abstract a set of attributes and functionalities common to classes

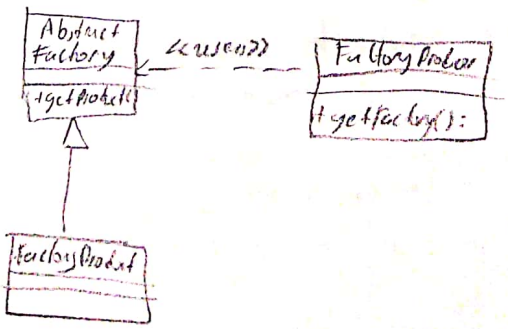
3 Constant data management: central repository of constant data, help to find easily and modify it in case of being necessary

4 Monitor:
- It's a way to design an application in a way that not produce unpredictable results when more than a thread try to access an object

Creational Patterns

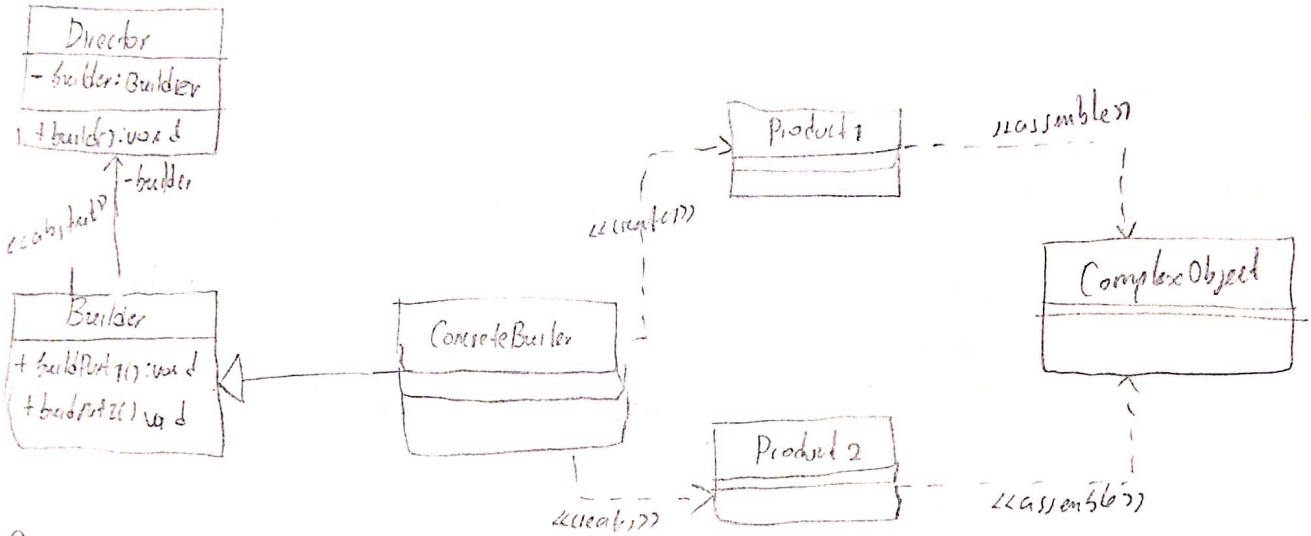
Provides ways to creat object while hiding the creation logic

1- Abstract factory → provides an interface to create families of related depend objects without specifying there concrete classes



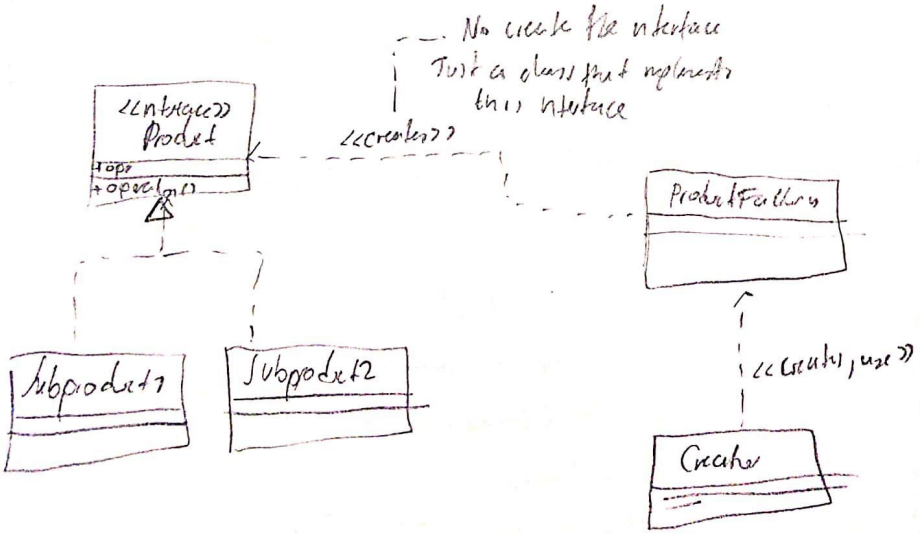
Builder

Separate the construction of a complex object from its representation so that the same construction process can create different representations



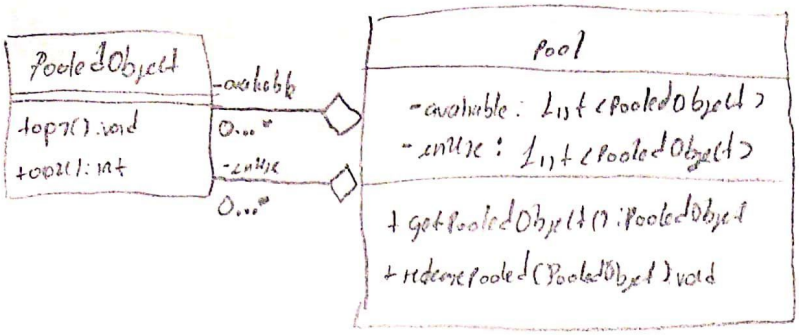
Factory Method

Define an interface to create an object



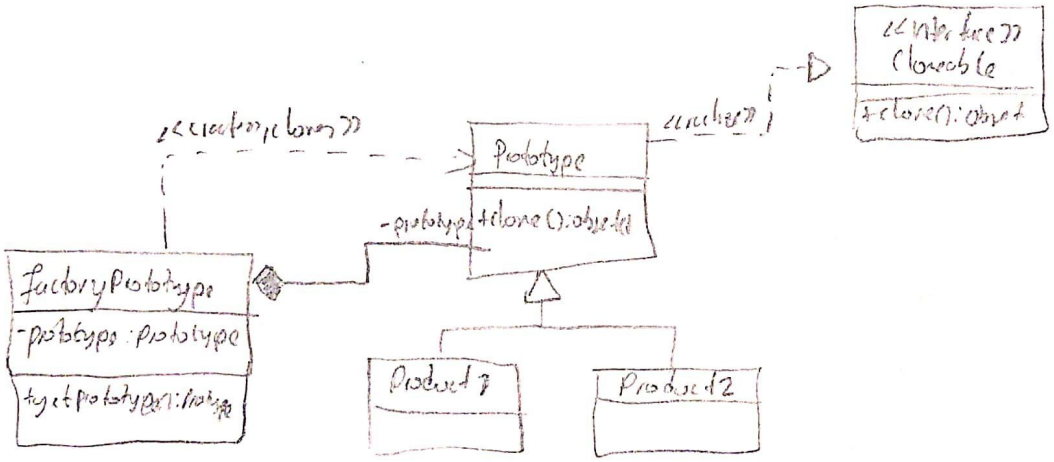
Object Pool

Offer a mechanism to kept ready a set of objects that are expensive to create



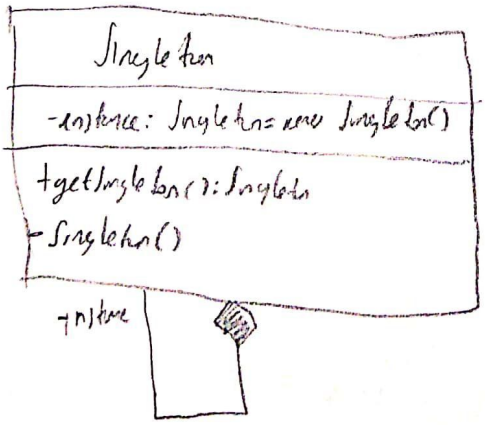
Prototype

Offers a mechanism to create objects using prototypical objects, this process is done by copying from prototypical instances.

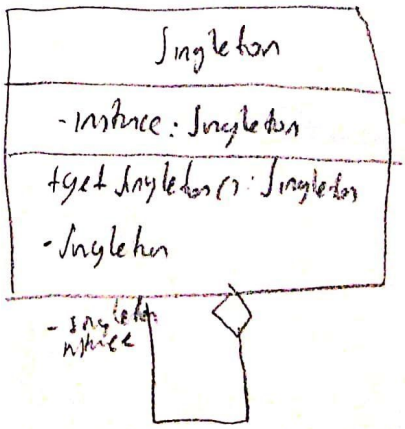


Singleton

Ensure a class has only one instance, and provide a global point of access to it



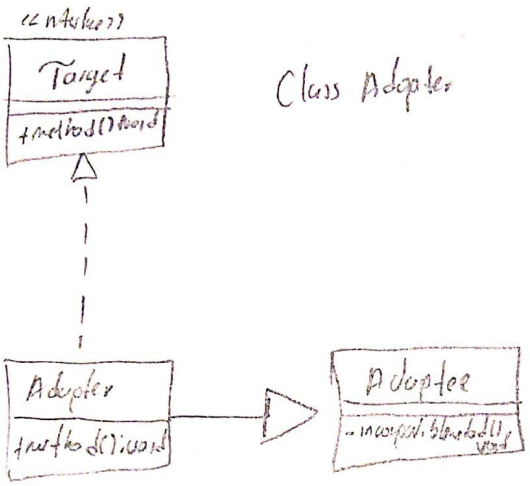
Eager Initialization



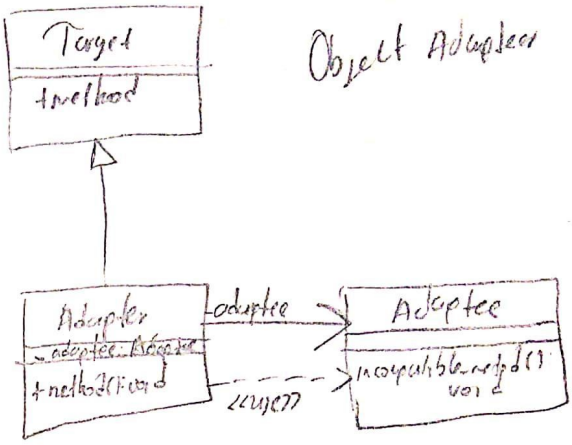
Lazy Initialization

Structural Patterns -> Deal with delegating responsabilities to other objects

Adapter -> Allows the conversion of the interface of class to another interface that client expects, Communicate incompatible interfaces

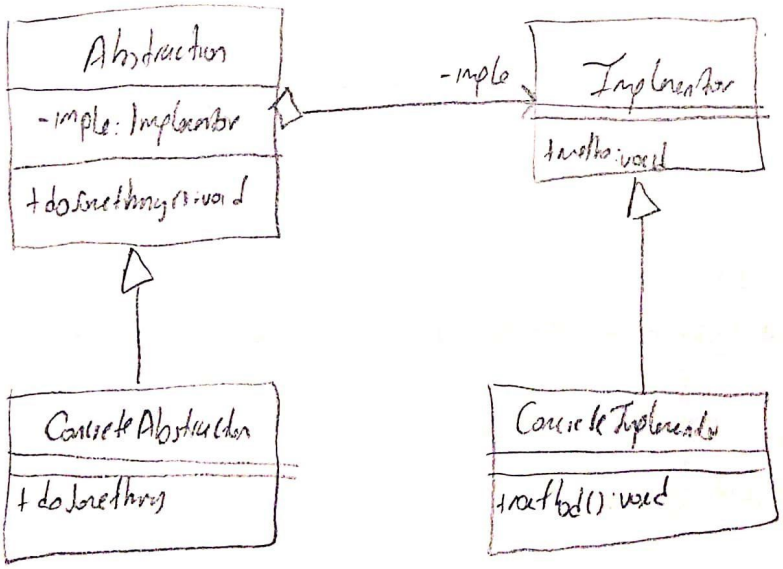


Class Adapter



Object Adapter

Bridge -> Allows the separation of an abstract interface from its implementation
-> Separate hierarchies of abstraction and implementation



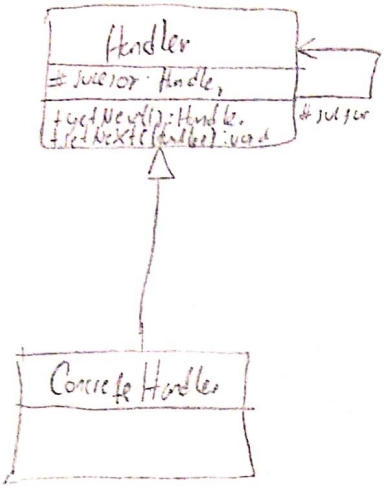
eg Message and Exception

- TextMessage
- VoiceMessage

- AES
- DES
- Choice

Chain of Responsability

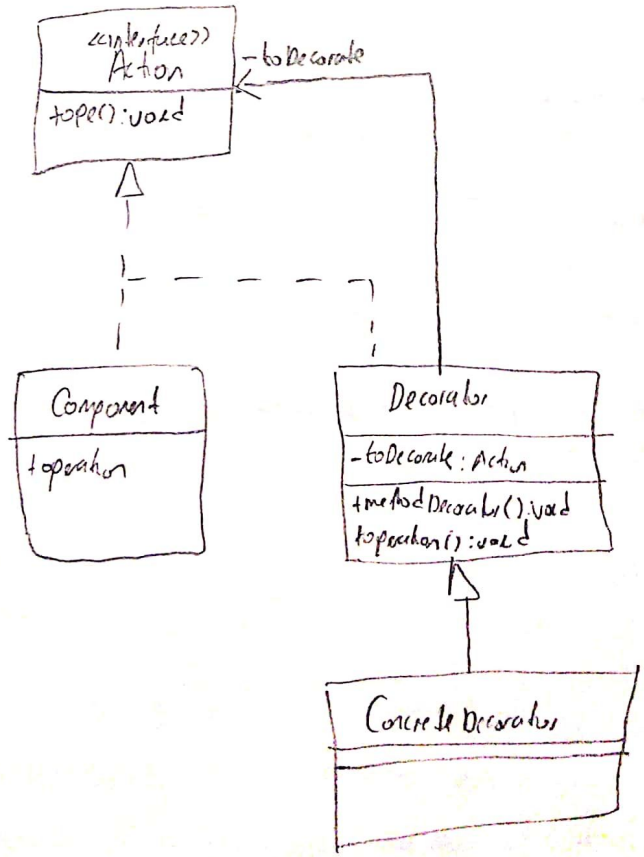
Avoid coupling a request object to a receiver object



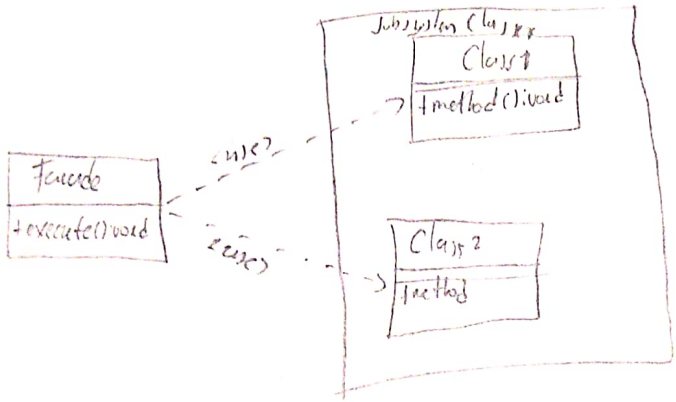
e.g : Authorize or sign some documents in a hierarchy of an enterprise

Decorator

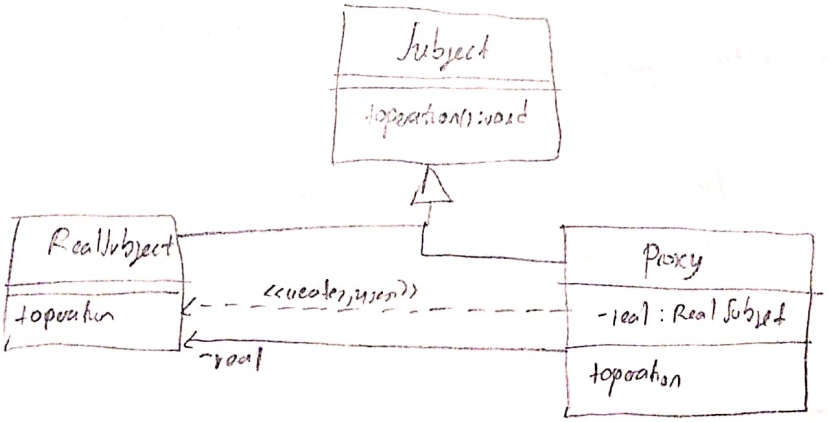
Extends the functionality of an object in a manner that is transparent to its clients without using inheritance



Facade → provide a higher level interface to a subsystem of classes, making the subsystem easier to use.



Proxy → Allows a separate object to be used as a substitute to provide controlled access to an object that is not accessible by normal means



~virtual proxy: helps to create objects on demand (until required)

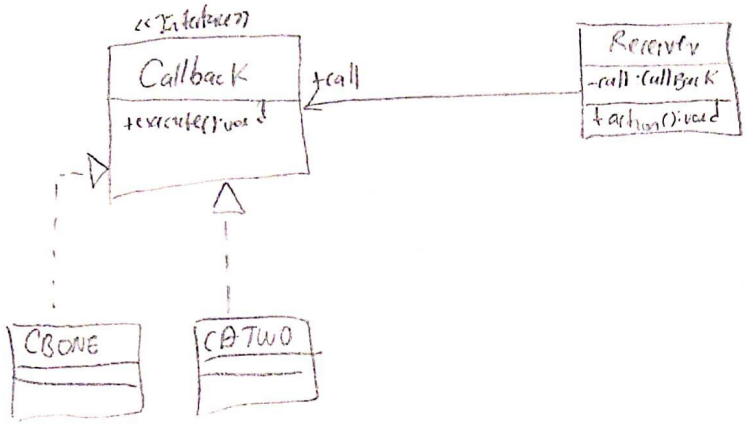
~(counting proxy: provide some kind of audit mechanism before executing an method on a target object

Behavioral Patterns

Deal with

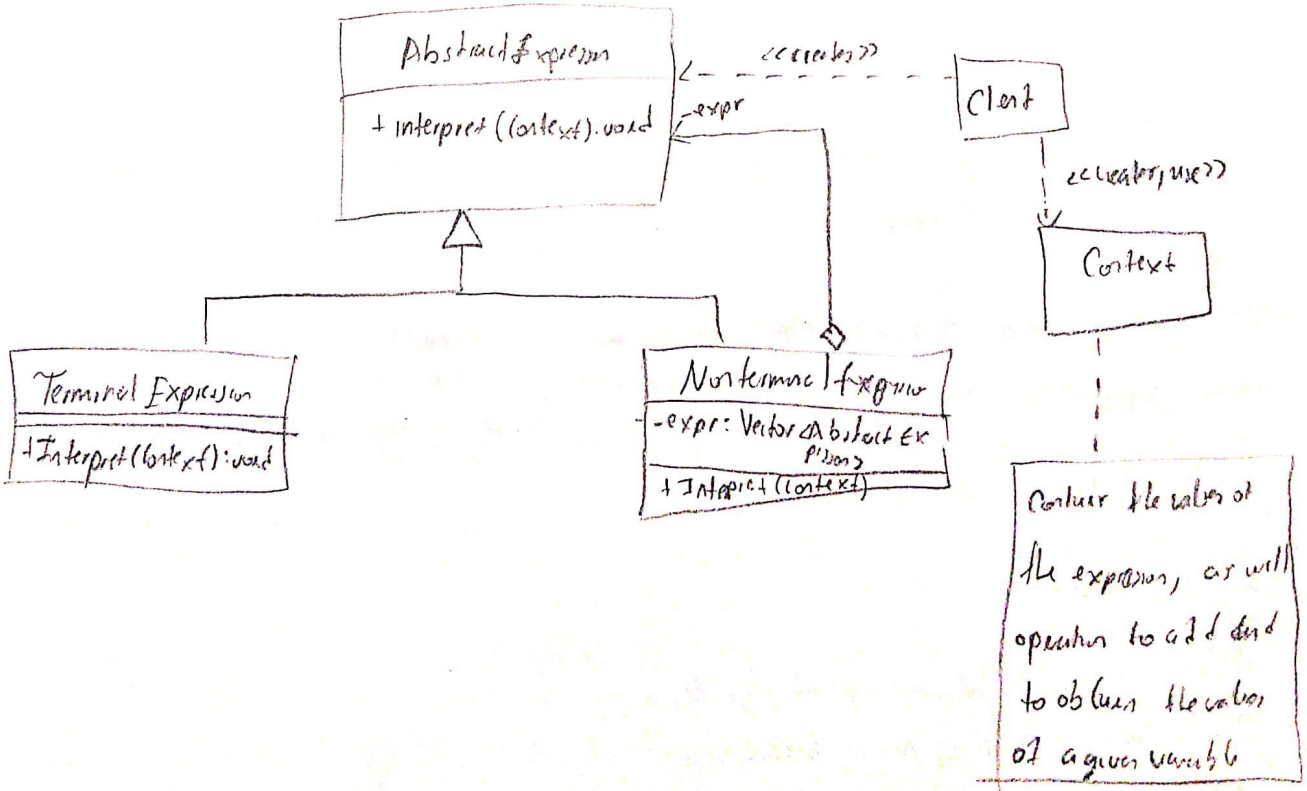
- assign responsibilities between different objects
- Describe the mechanism of communication between objects
- Define a mechanism for choosing different algorithms at runtime

Command → allow a request be encapsulated into a object, thereby letting you parameterize clients with different request.



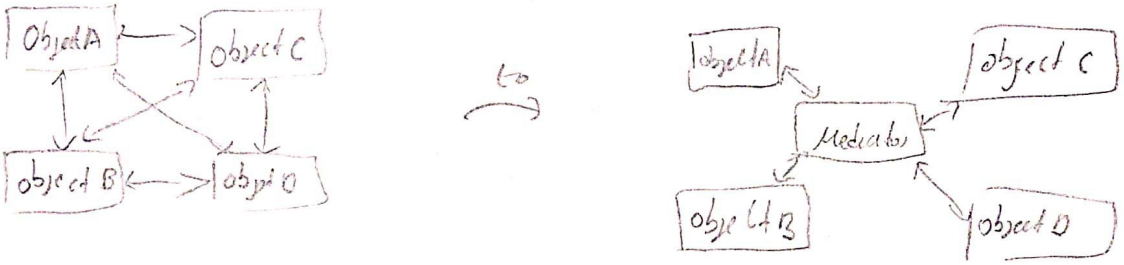
Interpreter

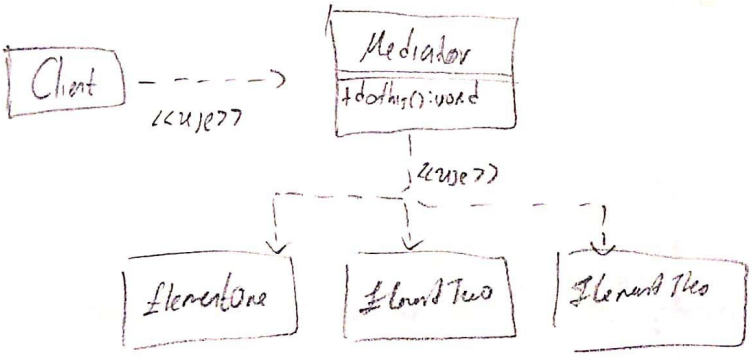
- Help to interpret operation using its grammar
- Given a language, define a representation for its grammar along with an interpreter that user the representation to interpret sentences in the language



(5+4)*(7+9)*(4+2)))

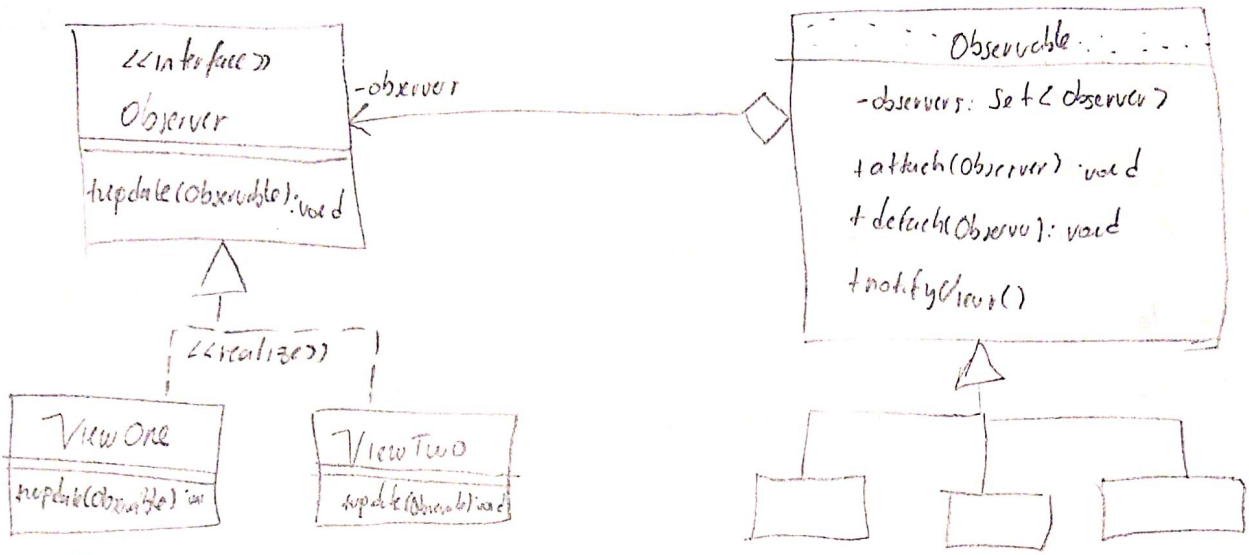
Mediator :- Encapsulates the direct Object Communication details among a set of objects, and
- Separate (mediator)
Define an object that encapsulates how a set of objects interact
- Promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently





Observer

Promotes a publisher-subscriber communication model when there is a one to many dependency between objects so that when one object change of state, all of its dependents are notified so they can update their state

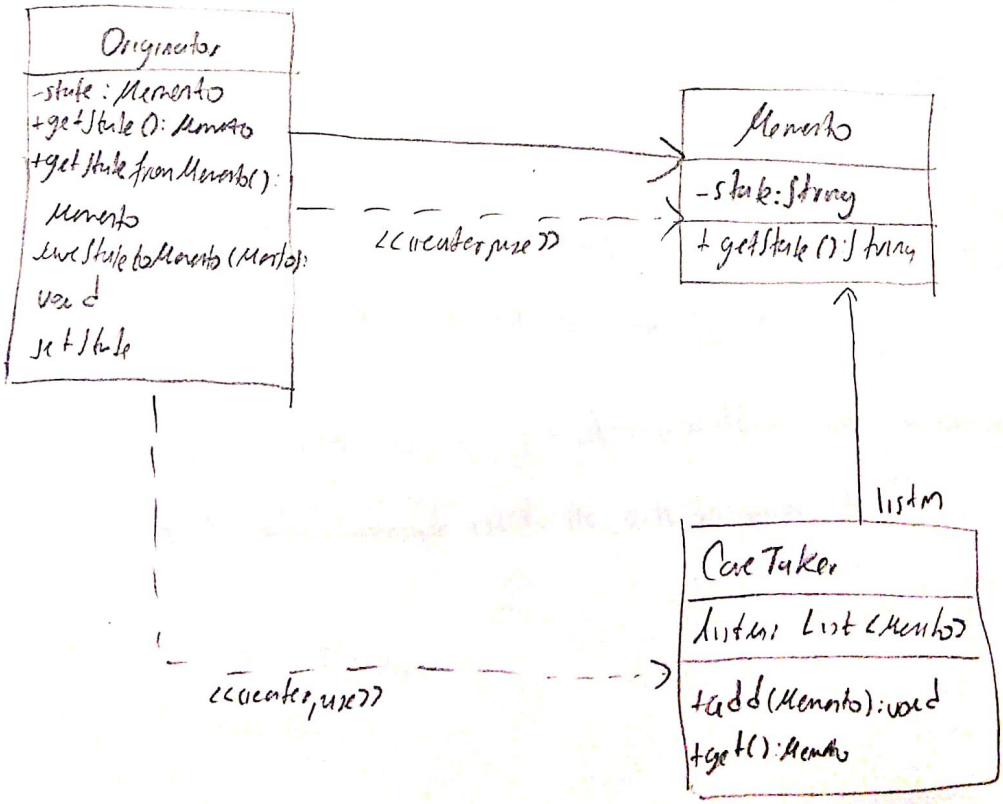


Idea?

We may unite observer an Mediator pattern, why? many objects interact each other, is that an enough reason

State

Allow the state of an object to be captured and stored

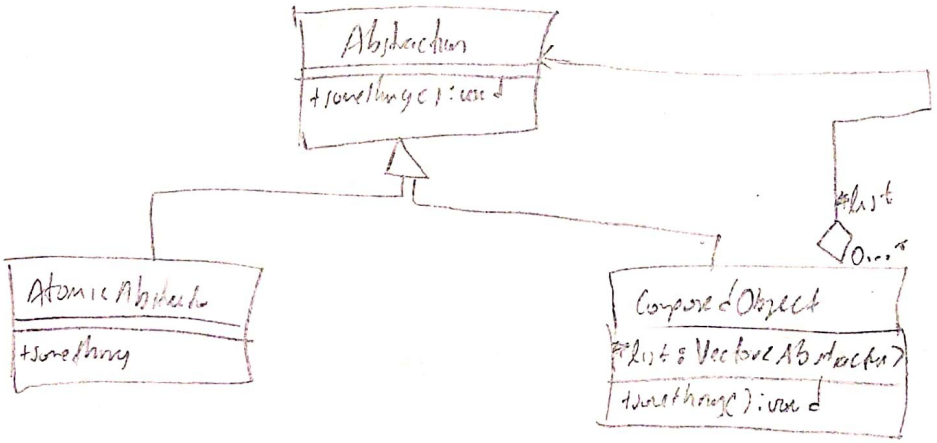


Collectional pattern

Deal with the details of how to compose classes and objects to form larger structures

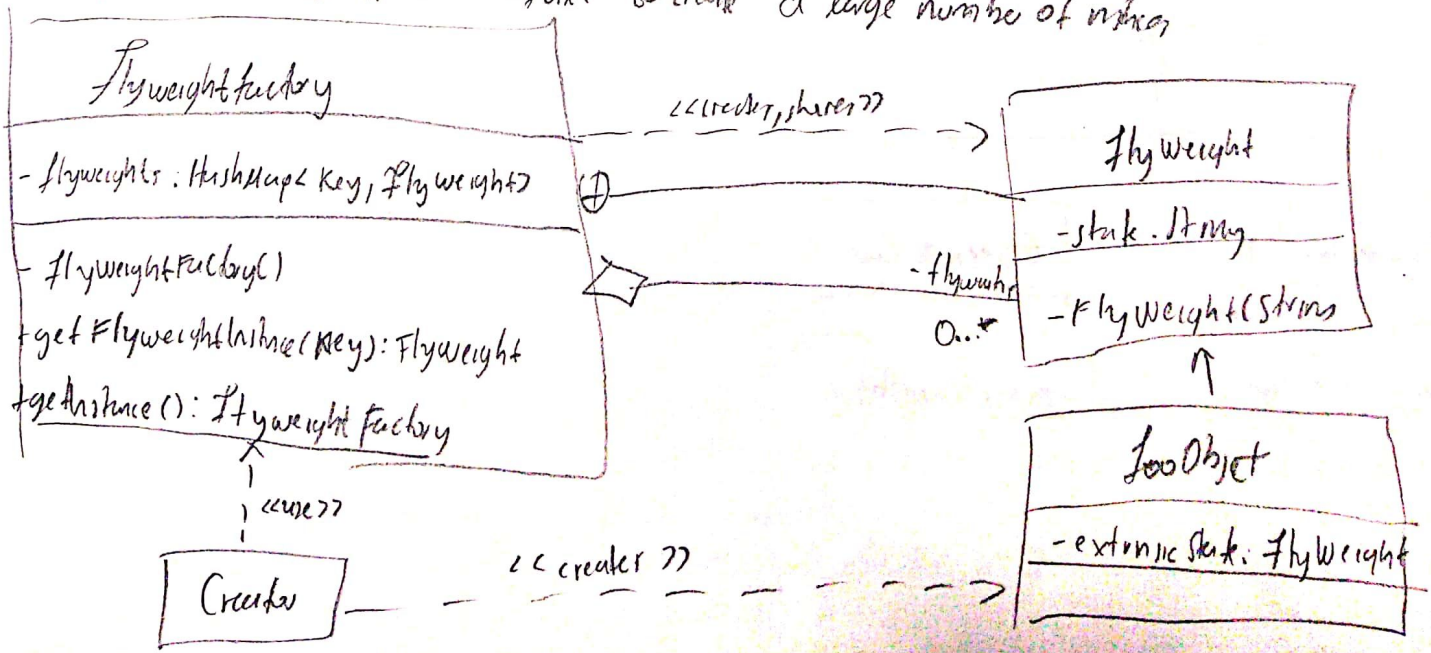
Composite

- Allow composite and individual objects treated uniformly
- Compose objects into tree structures to represent whole part hierarchies



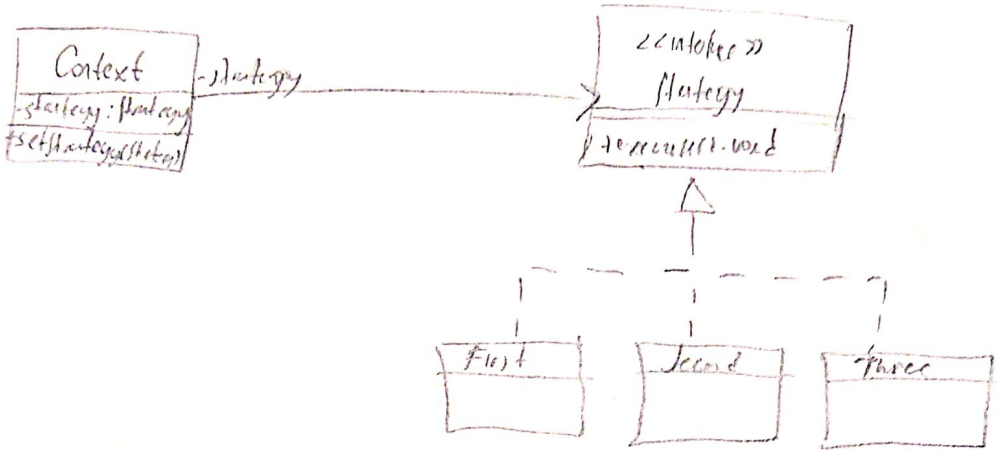
Flyweight

The invariant information are separated into two classes leading to savings in terms of memory usage and the amount of time required to create a large number of instances



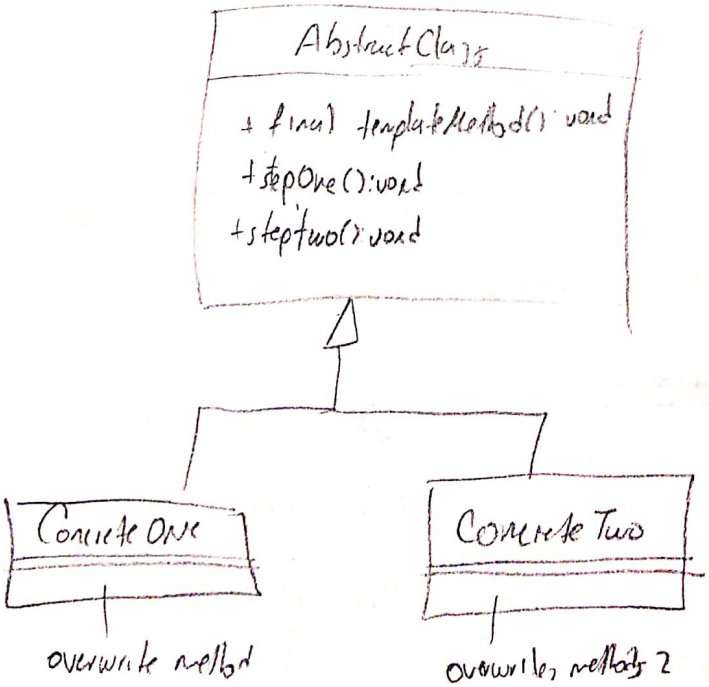
Strategy

Allows each of a family of related algorithms to be encapsulated into a set of different subclasses,



Template

Define the skeleton of an algorithm in an operation



Visitor

Allows to add a operation to be defined across a collection of different objects without changing the classes of objects on which it operates